What is Computational Complexity?

- Computational Complexity is a way to measure how much **time** and **memory (space)** an algorithm needs to solve a problem.

  It helps us understand how efficient and algorithm is, especially when the **input size becomes large.**

Types of Complexity:

1. Time Complexity:

- Measures how long an algorithm takes to complete.
- Depends on the number of operations based on input size.

2. Space Complexity:

- Measures how much memory **(RAM)** an algorithm needs to run.
- It's not about time, but about how much space is used during the program's execution.

Order of Computational Complexity (Best to Worst):

1. O (1) / Constant time (best& fastest): The algorithm takes the same amount of time regardless of the input size. Example: Array access by index.
2. O (log n) / Logarithmic time (fast): The time grows slowly as input size increases. Example: Binary Search.
3. O (n) / Linear time (acceptable): The time increases directly in proportion to the input size. Example: Searching in a list element by element.

4. O (n log n) / linear and logarithmic (commonly used in sorting): A mix of linear and logarithmic growth. Example: Efficient sorting algorithms.
5. O (n^2) / Quadratic time (slow): The time grows proportionally to the square of the input size. Example: Comparing each element with every other element in the array.
6. O (2^n) / Exponential time (very slow): The time doubles with each additional input element. Algorithms that try all possibilities. Example: Solving problems using brute force.
7. O (n^3) / Cubic time (slower): The time grows proportionally to the cube of the input size. Example: Triple nested loops.
8. O (n!) / Factorial time (worst& slowest): The time grows extremely fast with input size / very inefficient. Example: Generating all possible permutations of a string.